# Design Patterns in Python

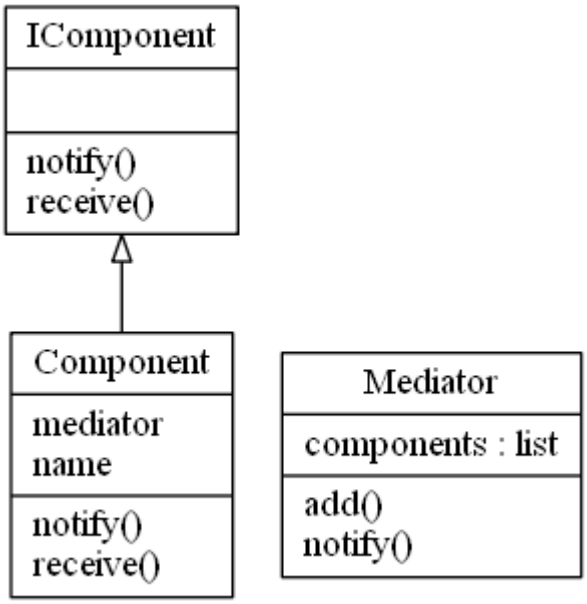
## Mediator Design Pattern

### Description

The mediator pattern is a behavioural pattern that defines an object that encapsulates how a set of objects interact.

With the mediator pattern, communication between objects is encapsulated within a mediator object.

Objects communicate through the mediator rather than directly with each other.




### Source Code

**mediator.py**

```
from abc import ABCMeta, abstractmethod


class IComponent(metaclass=ABCMeta):
    @staticmethod
    @abstractmethod
    def notify(msg):
        """The required notify method"""

    @staticmethod
    @abstractmethod
    def receive(msg):
        """The required receive method"""


class Component(IComponent):
    def __init__(self, mediator, name):
        self.mediator = mediator
```

```
        self.name = name

    def notify(self, message):
        print(self.name + ": >>> Out >>> : " + message)
        self.mediator.notify(message, self)

    def receive(self, message):
        print(self.name + ": <<< In <<< : " + message)


class Mediator():
    def __init__(self):
        self.components = []

    def add(self, component):
        self.components.append(component)

    def notify(self, message, component):
        for _component in self.components:
            if _component != component:
                _component.receive(message)


MEDIATOR = Mediator()
COMPONENT1 = Component(MEDIATOR, "Component1")
COMPONENT2 = Component(MEDIATOR, "Component2")
COMPONENT3 = Component(MEDIATOR, "Component3")
MEDIATOR.add(COMPONENT1)
MEDIATOR.add(COMPONENT2)
MEDIATOR.add(COMPONENT3)

COMPONENT1.notify("data")
```

Copyright © Sean Bradley 2019 : Page 2 / 2